(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 2 096 544 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
02.09.2009 Bulletin 2009/36

(51) Int Cl.:
G06F 11/14 *(2006.01)*

(21) Application number: **09151447.1**

(22) Date of filing: **27.01.2009**

(84) Designated Contracting States:
AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HR HU IE IS IT LI LT LU LV MC MK MT NL NO PL PT RO SE SI SK TR
Designated Extension States:
AL BA RS

(30) Priority: **04.02.2008 US 25660**

(71) Applicant: **Honeywell International Inc.**
Morristown, NJ 07962 (US)

(72) Inventors:
- **Ramegowda, Yogesha Aralakuppe**
  Morristown, NJ 07962-2245 (US)
- **Dangeti, Srinivasa R.**
  Morristown, NJ 07962-2245 (US)
- **Chopra, Puja**
  Morristown, NJ 07962-2245 (US)
- **Pesala, Narasimha Rao**
  Morristown, NJ 07962-2245 (US)
- **Gautam, Puri**
  Morristown, NJ 07962-2245 (US)
- **Kop, Shruti**
  Morristown, NJ 07962-2245 (US)
- **Raj, Darshan**
  Morristown, NJ 07962-2245 (US)
- **Sivaraman, Mani**
  Morristown, NJ 07962-2245 (US)
- **Puppala, Yugandhar Kumar**
  Morristown, NJ 07962-2245 (US)
- **Muthusamy, Kaarthikeyan**
  Morristown, NJ 07962-2245 (US)
- **Jethe, Sachin**
  Morristown, NJ 07962-2245 (US)
- **Suresh, Mugdalbetta Rajesh**
  Morristown, NJ 07962-2245 (US)

(74) Representative: **Buckley, Guy Julian**
Patent Outsourcing Limited
1 King Street
Bakewell
Derbyshire DE45 1DZ (GB)

(54) System and method for detection and prevention of flash corruption

(57) A non-volatile memory device comprises an application code sector of sufficient size to store a first copy of an application code and a second copy of the application code; and a boot sector having a boot loader code embodied therein. The boot loader code is configured to cause a processor to check the integrity of both the first and second copies of the application code; if the first copy is corrupted, overwrite the first copy of the application code with the second copy; and if the second copy is corrupted, overwrite the second copy of the application code with the first copy.

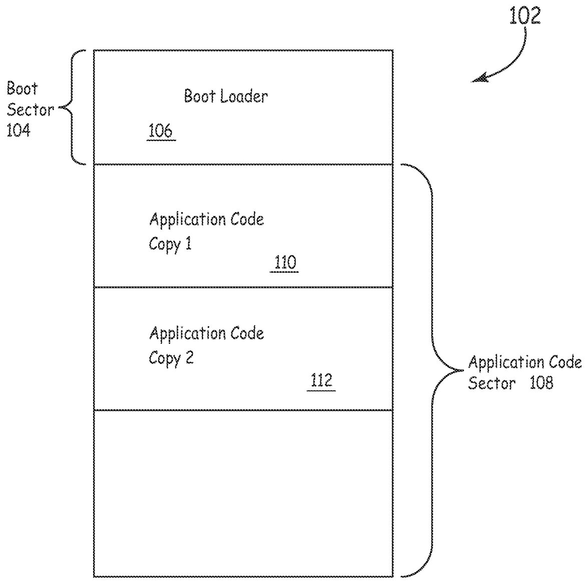


FIG. 1

Printed by Jouve, 75001 PARIS (FR)

Description

BACKGROUND

[0001] Microcontrollers in an embedded system typically include a central processing unit (CPU), non-volatile memory (such as EEPROM or flash memory), interfaces, random access memory (RAM), and other peripherals integrated onto a single integrated circuit. Hence, the number of chips, wires, and space needed is reduced compared to using separate chips. In addition, unlike general purpose microprocessors, microcontrollers are typically designed to carry out specific functions which increases their cost-effectiveness.

[0002] However, microcontrollers are vulnerable to data corruption such as corruption due to code run-away. Code run-away can be caused by faulty code, operating the Micro-Controller Unit (MCU) outside its specification or by a major electromagnetic interference (EMI) or electrical noise event. By definition, it is not well defined what will happen during code run-away, but it is caused by the out-of-specification operating environment effectively corrupting the program counter resulting in the MCU behaving unpredictably. A corrupted program counter could lead to a jump to programming code that performs the flash erase or write operation, resulting in accidental corruption of flash memory data that contains application code. Once the application code is corrupted, it is typically not possible to recover until the correct application code is programmed again.

[0003] For the reasons stated above, and for other reasons stated below which will become apparent to those skilled in the art upon reading and understanding the present specification, there is a need in the art for a system and method to detect and recover from flash corruption.

SUMMARY

[0004] The above mentioned problems and other problems are resolved by the present invention and will be understood by reading and studying the following specification.

[0005] In one embodiment, a non-volatile memory device is provided. The non-volatile memory device comprises an application code sector of sufficient size to store a first copy of an application code and a second copy of the application code; and a boot sector having a boot loader code embodied therein. The boot loader code is configured to cause a processor to

check the integrity of both the first and second copies of the application code; if the first copy is corrupted, overwrite the first copy of the application code with the second copy; and if the second copy is corrupted, overwrite the second copy of the application code with the first copy.

DRAWINGS

[0006] Features of the present invention will become apparent to those skilled in the art from the following description with reference to the drawings. Understanding that the drawings depict only typical embodiments of the invention and are not therefore to be considered limiting in scope, the invention will be described with additional specificity and detail through the use of the accompanying drawings, in which:

Figure 1 is a block diagram of a non-volatile memory device according to one embodiment of the present invention.

Figure 2 is a block diagram of a microcontroller according to one embodiment of the present invention.

Figure 3 is a flow chart of a method of detecting and recovering from corrupted data in a non-volatile memory device according to one embodiment of the present invention.

Figure 4 is a flow chart of a method of implementing a non-volatile memory device to prevent data corruption according to one embodiment of the present invention.

[0007] In accordance with common practice, the various described features are not drawn to scale but are drawn to emphasize specific features relevant to the present invention. Like reference numbers and designations in the various drawings indicate like elements.

DETAILED DESCRIPTION

[0008] In the following detailed description, reference is made to the accompanying drawings that form a part hereof, and in which is shown by way of illustration specific illustrative embodiments in which the invention may be practiced. These embodiments are described in sufficient detail to enable those skilled in the art to practice the invention, and it is to be understood that other embodiments may be utilized and that logical, mechanical, and electrical changes may be made without departing from the scope of the present invention. Furthermore, the method presented in the drawing figures or the specification is not to be construed as limiting the order in which the individual steps may be performed. The following detailed description is, therefore, not to be taken in a limiting sense.

[0009] Embodiments of the present invention enable the detection and recovery of corrupted data in a non-volatile memory device, such as a Flash memory. In particular, embodiments of the present invention enable more robust detection and recovery mechanism of corrupted application code than conventional non-volatile memory devices through the use of multiple copies of the application code and modified functionality of a boot loader code also stored on the non-volatile memory device.

[0010] Figure 1 is a block diagram of a non-volatile memory device 102 according to one embodiment of the present invention. Examples of non-volatile memory devices include, but not limited to, EEPROM and Flash memory devices. Device 102 comprises a boot sector 104 and an application code sector 108. A boot sector is defined, as used herein, as a section of memory device 102 used for storing a boot loader code. Hence, boot sector 104 contains boot loader code 106. Similarly, an application code sector 108 is defined, as used herein, as a section of memory device 102 used for storing an application code. Application code is a program which directly applies the capabilities of a microcontroller to perform a specific task. In embodiments of the present invention, application code sector 108 is of sufficient size to store two complete copies of an application code. Hence, application code sector 108 contains a first copy 110 of an application code and a second copy 112 of the same application code.

[0011] Boot loader code 106 stored on boot sector 104 is configured to cause a processor, such as processor 214 described below, executing the boot loader code 106 to check the integrity of first copy 110 and second copy 112 of the application code. In particular, the integrity of the first copy 110 and the second copy 112 is checked on each hardware and/or software reset. Hardware resets (also known as hard boots) involve removing power from memory device 102 and subsequently restoring power (e.g. a power cycle). Additionally, hardware resets also include resets in which a system is restarted without performing any shut-down procedures. A software reset (also referred to as a soft boot or warm boot) is a reset under software control without completely removing power from non-volatile memory device 102. Software resets typically include an ordered restart procedure. When either type of reset occurs, control of the processor is passed back to boot loader code 106. Boot loader code 106 then causes the processor to again check the integrity of the first copy 110 and the second copy 112.

[0012] In some embodiments, checking the integrity of the first copy 110 and the second copy 112 includes performing a Cyclic Redundancy Check (CRC) on the first copy 110 and second copy 112. However, it is to be understood that other known techniques of checking integrity can be used in other embodiments. If the first copy 110 fails the integrity check but the second copy 112 passes, boot loader code 106 causes the processor to overwrite the first copy 110 with the second copy 112. Similarly, if the second copy 112 fails the integrity check but the first copy 110 passes, boot loader code 106 causes the processor to overwrite the second copy 112 with the first copy 110. In this manner, errors in either copy are corrected with the other good copy. In some embodiments, boot loader code 106 performs an additional integrity check if either copy was overwritten to ensure the copy was overwritten successfully. Boot loader code 106 then transfers control to one of the copies.

[0013] In the event that both the first copy 110 and the second copy 112 fail the integrity check, boot loader code 106 does not cause the processor to transfer control to either copy. Instead, boot loader code 106 logs a CRC fault and retains control while waiting for an external command. Discuss - Integrity check to be performed on the overwritten copy before transferring the control.

[0014] If both first copy 110 and second copy 112 pass the integrity check, boot loader code 106 causes the processor to transfer control to one of the copies based on a pre-determined default. For example, in one embodiment, if both copies pass the integrity check, control is transferred to first copy 110 by default. If only one of the copies passes the integrity check, control is transferred to the copy which passed the integrity check. In some embodiments, both first copy 110 and second copy 112 are configured to cause the processor to continue to perform integrity checks as a background process along with its normal functionality. The integrity checks are run on both copies, in some embodiments, regardless of which copy is currently being executed by the processor. In other embodiments, the integrity check is only performed on the copy with control. If the copy with control fails the integrity check, the copy logs the fault and forces a software reset. The software reset results in transferring control back to boot loader code 106 which again checks the integrity of first copy 110 and second copy 112 as described above.

[0015] Figure 2 is a block diagram of a microcontroller 200 which implements a non-volatile memory device 202 according to embodiments of the present invention. Device 202 is similar to device 102 described above. Microcontroller 200 microcontroller 200 is integrated onto a single chip and also comprises a random access memory (RAM) 216, a processor 214, and input/output ports 218. Microcontroller 200 may also contain other peripherals 220, such as a timer module, analog-to-digital converter, etc. as known to one of skill in the art.

[0016] Input/output ports 218 provide signals from/to other devices to/from microcontroller 200, such as user input devices, sensors, etc. Processor 214 processes signals received over input/output ports 218. In processing signals, processor 214 uses RAM 216 to store dynamic data used by processor 214, such as data received from input/output ports 218 and code from non-volatile memory 202.

[0017] In operation, on each hardware or software reset, processor 200 executes a boot loader code (such as boot loader code 106 in Fig. 1). The boot loader code causes the processor to check the integrity of a first copy and a second copy of application code stored on the non-volatile device 202 as described above. Thus, prior to execution of the application code, the integrity of each copy is checked and corrected. Such action prevents faulty application code from being executed on startup or after a reset. In addition, each copy of the application code is also configured to cause processor 214 to continue to check the integrity of the first and second copies as a background process. As described above, if the copy being executed does not pass the integrity check, a software reset is forced to pass control back to the boot loader code.

[0018] Figure 3 is a flow chart depicting a method 300 of detecting and recovering from corrupted data in a non-volatile memory device, such as memory device 102, according to one embodiment of the present invention. At 302, the integrity check of a first copy and a second copy of an application code is performed. In particular, in this example a CRC is performed on both the first and second copies, as described above. In embodiments of the present invention, the initial integrity check occurs prior to transferring control of a processor to one of the first and second copies. As described above, a boot loader code is used in some embodiments to cause the processor to perform the integrity check.

[0019] At 304, it is determined if the first copy passed the integrity check. If the first copy did not pass the integrity check, it is determined at 306 if the second copy passed the integrity check. If the second copy did not pass the integrity check at 306, control is retained by the boot loader code, at 308, to wait for an external command as described above. For example, an external command is a command from a user or other device. If the second copy does pass the integrity check at 306, the faulty first copy is overwritten with the second copy at 312.

[0020] If the first copy does pass the integrity check at 304, it is determined if the second copy passed the integrity check at 310. If the second copy did not pass the integrity check at 310, the faulty second copy is overwritten with the first copy at 312. If the second copy also passed the integrity check at 310, control is transferred to one of the copies at 314. If both copies passed the integrity check, control is transferred to one of the copies as a default. For example, in one embodiment, by default, control is transferred to the first copy if both copies passed the integrity check.

[0021] If one of the copies is overwritten at 312, an optional integrity check is performed at 313 to determine if the overwrite was successful and to ensure that the copies pass the integrity check before transferring control. If the copies do not pass the integrity check at 313, control is retained at 308 as described above. If the copies do pass the integrity check at 313, control is transferred to one of the copies at 314. For example, in one embodiment, control is transferred to the second copy if the first copy was overwritten with the second copy passed at 312. Alternatively, control is passed to the first copy if the second copy did not pass the integrity check at 310.

[0022] Once control is transferred to one of the copies at 314, an integrity check is performed as a background process during execution of the copy with control at 316. In particular, each of the first copy and the second copy are configured to cause the processor to perform an integrity check on each copy. If the copy with control passes the integrity check at 318, method 300 returns to 316 where the integrity continues to be checked as a background process. If the copy with control did not pass the integrity check at 318, the processor logs the fault at 320 and forces a software reset at 322. The software reset will cause control to be returned to the boot loader code which again checks the integrity of the first and second copies at 302. Therefore, errors or corruption in either copy of the application code is detected and corrected through method 300. In addition to the detection and correction provided by method 300, in some embodiments, the boot loader code is locked or secured to prevent flash corruption. Locking or securing the boot loader code prevents changes to the boot loader code using normal write or erase commands. One manner of locking the boot loader code is described in fig. 4.

[0023] Figure 4 is a flow chart depicting a method of implementing a non-volatile memory device, such as device 102, to prevent data corruption according to one embodiment of the present invention. At 402, boot loader code is loaded onto boot sector of the non-volatile memory device. In particular, the boot loader code is loaded as part of the manufacturing process. At 404, the boot loader code is secured by setting bits in the protection register which correspond to the boot loader code. For example, in one embodiment, the protection register bits are set by an external system which loads the boot loader code onto the non-volatile memory device. In another embodiment, the boot loader code is configured to set the bits when executed.

[0024] At 406, the first and second copies of the application code are loaded onto the application code sector of the non-volatile memory device. At 408, it is periodically determined if a new baseline or released version of the boot loader code is available. If a new version is available, the boot loader code is unsecured, at 410, by erasing the memory using one of a Background Debug Module (BDM), JTAG or chip erase commands. Method 400 then returns to 402 where the new version of the boot loader code is loaded onto the boot sector of the non-volatile memory device. If a new version of the boot loader code is not available at 408, method 400 ends at 412.

[0025] Although specific embodiments have been illustrated and described herein, it will be appreciated by those of ordinary skill in the art that any arrangement, which is calculated to achieve the same purpose, may be substituted for the specific embodiment shown. For example, although the exemplary embodiments described above discuss two copies of the application code, it is to be understood that additional copies can be used in other embodiments. This application is intended to cover any adaptations or variations of the present invention. Therefore, it is manifestly intended that this invention be limited only by the claims and the equivalents thereof.

Claims

1. A method (300) of detecting and recovering from corrupted data in a non-volatile memory device, the method comprising:

checking the integrity of a first copy of an application code stored in the non-volatile memory device (302);
checking the integrity of a second copy of the application code stored in the non-volatile memory device (302);
if the first copy fails the integrity check and the second copy passes the integrity check, overwriting the first copy with the second copy (304, 306, 312) and transferring control of a processor from a boot loader code to the second copy (314); and
if the first copy passes the integrity check and the second copy fails the integrity check, overwriting the second copy with the first copy (304, 310, 312) and transferring control of the processor from the boot loader code to the first copy (314).

2. The method of claim 1, wherein checking the integrity of the first and second copies comprises performing a Cyclic Redundancy Check (CRC) on the first and second copies (302).

3. The method of claim 1, further comprising:

if both the first and second copies fail the integrity check, maintaining control of the processor with the boot loader code (304, 306, 308).

4. The method of claim 1, further comprising:

setting one or more bits in a protection register corresponding to the boot loader code to prevent changes to the boot loader code (404).

5. The method of claim 1, further comprising
checking the integrity of the first and second copies after transferring control to one of the first and second copies (316); and
if the copy with control of the processor fails the integrity check, forcing a software reset to transfer control back to the boot loader code (322).

6. The method of claim 1, wherein checking the integrity of the first and second copies comprises checking the integrity of the first and second copies on every hardware and software reset (302, and paragraph [0021]).

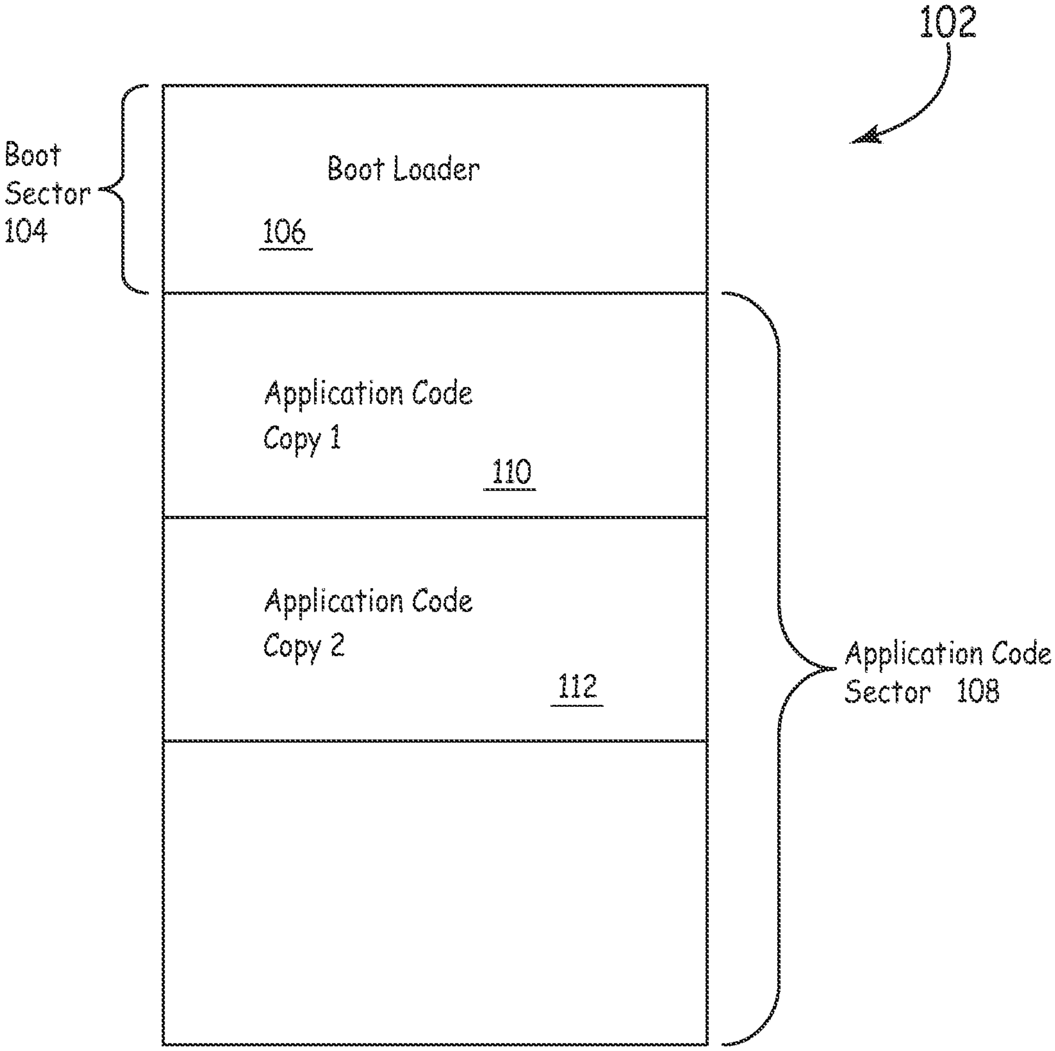

102
Boot Sector 104
Boot Loader
106
Application Code Copy 1
110
Application Code Copy 2
112
Application Code Sector 108

FIG. 1

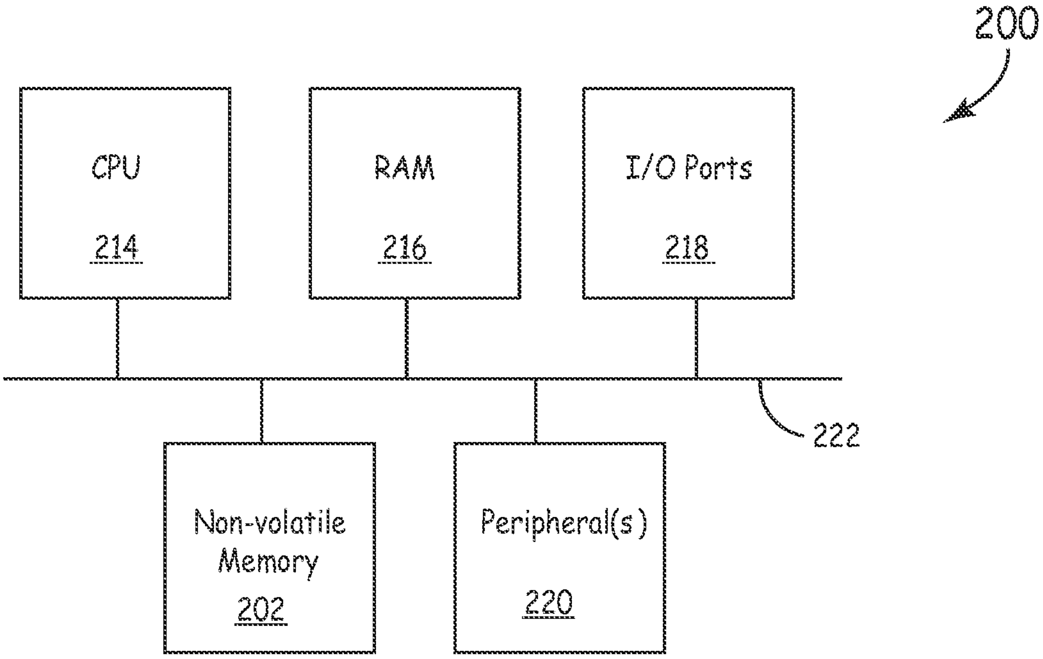
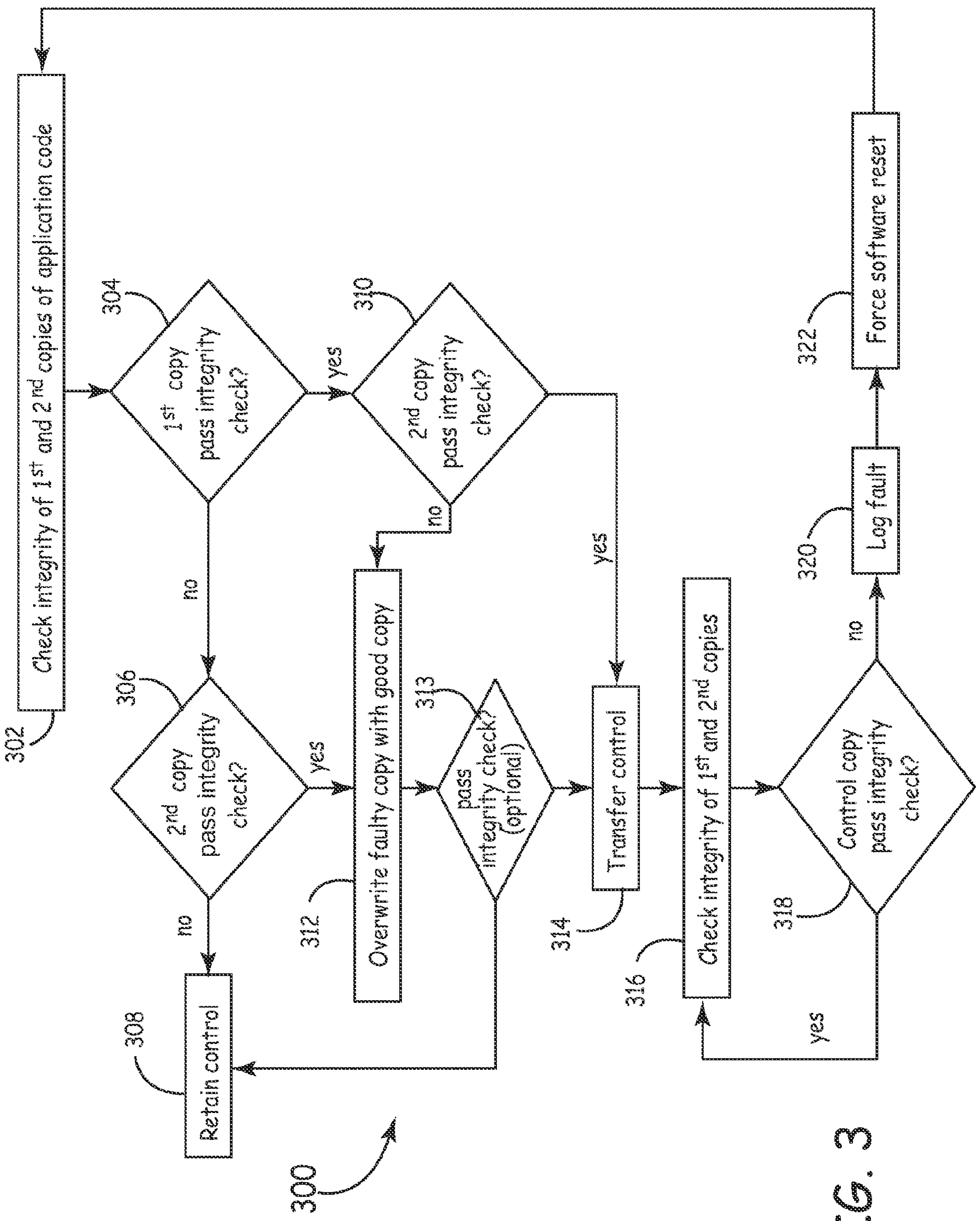

200
CPU
214
RAM
216
I/O Ports
218
222
Non-volatile Memory
202
Peripheral(s)
220

FIG. 2



302
Check integrity of 1st and 2nd copies of application code
304
1st copy pass integrity check?
yes
no
310
2nd copy pass integrity check?
no
yes
306
2nd copy pass integrity check?
yes
no
308
Retain control
312
Overwrite faulty copy with good copy
313
pass integrity check? (optional)
314
Transfer control
316
Check integrity of 1st and 2nd copies
318
Control copy pass integrity check?
yes
no
320
Log fault
322
Force software reset
300

FIG. 3

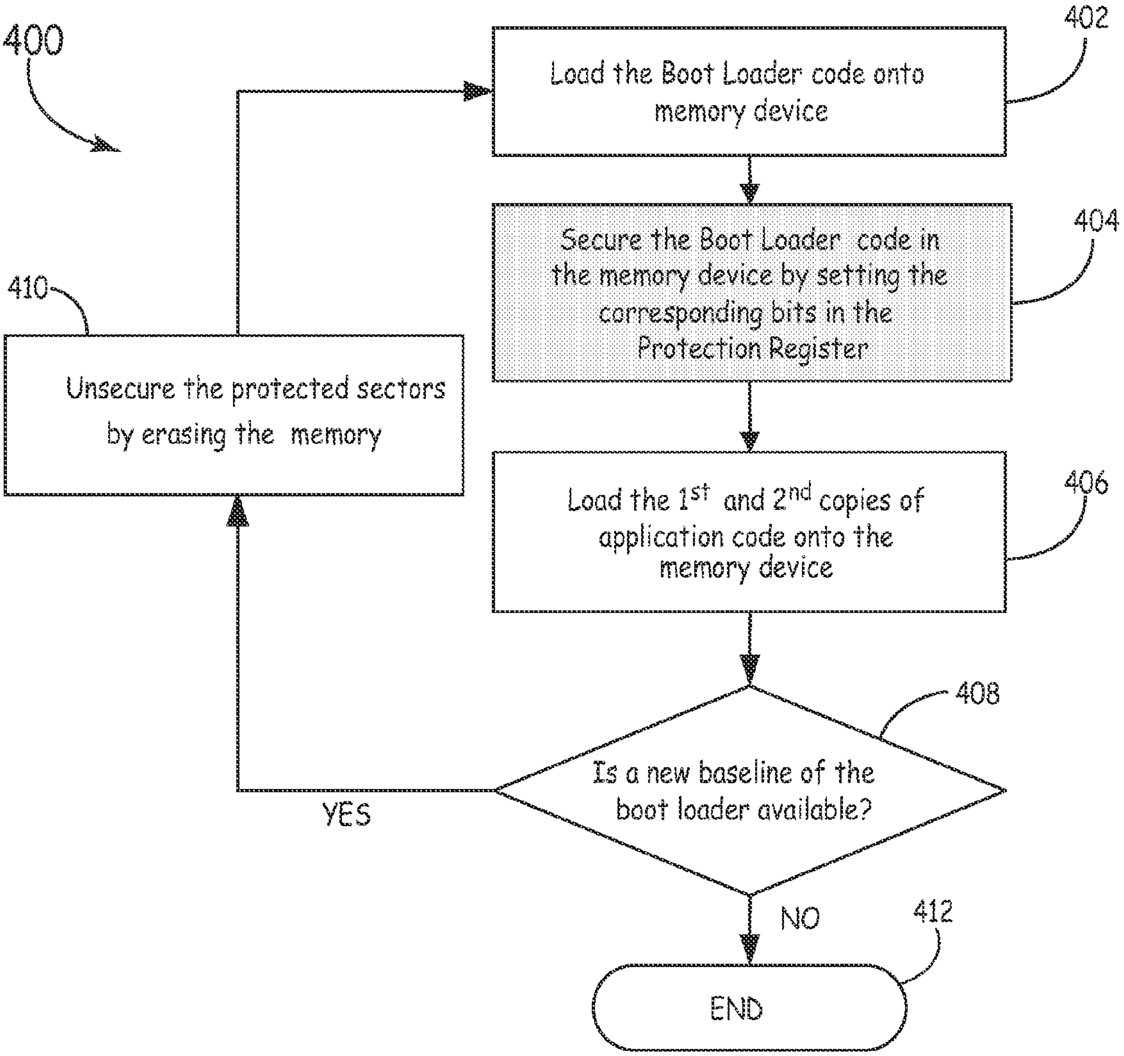

400
Load the Boot Loader code onto memory device
402
Secure the Boot Loader code in the memory device by setting the corresponding bits in the Protection Register
404
Load the 1st and 2nd copies of application code onto the memory device
406
Is a new baseline of the boot loader available?
408
YES
Unsecure the protected sectors by erasing the memory
410
NO
412
END

FIG. 4